CSE 4111/5111/6111 Computability
Jeff Edmonds
Assignment 3: Diagonalization & Halting Problem
Due: One week after shown in slides

First Person:
- Family Name:
- Given Name:
- Student #:
- Email:

Second Person:
- Family Name:
- Given Name:
- Student #:
- Email:

Guidelines:

- You are strongly encouraged to work in groups of two. Do not get solutions from other pairs. Though you are to teach & learn from your partner, you are responsible to do and learn the work yourself. Write it up together. Proofread it.
- Please make your answers clear and succinct. helpful hints.
- Relevant Readings:
  - 02-Computable Slides from the lectures
  - Cooks (pg 71) notes
- This page should be the cover of your assignment.

Problem Name	Max Mark	
1 Rationals	10	
2 Power Sets	20	
3 Describable Reals	10	
Total	40	

1. Rationals:

   (a) Each fraction has an infinite description, eg $\frac{1}{3} = 0.33333\ldots$. Didn't we say that this means the set of fractions $\mathcal{Q}$ is uncountable? Explain why or why not.

   (b) Study the prove that the set of real numbers is uncountable. Use the exact same proof to show that $\mathcal{Q}$ is uncountable. What if anything goes wrong in the proof?

2. Power Sets: Let U be a set of objects. In this question we will first have it be the set of positive integers and then be the set of positive reals less than one. The power set of U is the set of all subsets of U. If U is finite, then its power set has cardinality $2^{|U|}$ elements. Hence, power set of U is often denoted by $2^U = \{s \mid s \subseteq U\}$. Similarly, denote $2^U_{finite} = \{s \mid s \subseteq U \text{ and } |s| \text{ is finite}\}$. We will consider the relative sizes of U, 2^U_{finite}, and 2^U.

   (a) Recall that we say $|2^U_{finite}| \leq |U|$ if $\exists$ a function $F : 2^U_{finite} \rightarrow U$ such that $\forall s \in 2^U_{finite}, F(s) \in U$ and $\forall s, s' \in 2^U_{finite},\ s \neq s' \Rightarrow F(s) \neq F(s')$. A common way to prove $s \neq s' \Rightarrow F(s) \neq F(s')$ is to provided the inverse function F^{-1} and prove that $\forall s\ F^{-1}(F(s) = s$. (Though I did want you think about this, I don't ask you to do it.)

      i. Let $U = \mathcal{N}$ denote the set of positive integers. Then $2^{\mathcal{N}}_{finite}$ denotes the set of finite subsets of $\mathcal{N}$. Prove that $2^{\mathcal{N}}_{finite}$ is countable by defining a concrete function $F(s) = u_s$ mapping each finite subsets s of the positive integers to a unique integer u_s. Use the ascii technique given in the slides.
      - If $s = \{24, 8\}$, what is $F(s)$?
      - What happens if you don't encode the commas?
      - Does the fact that the integers in s can be put into different orders create a problem?
      - What two properties of s are key in proving that for every $s \in 2^{\mathcal{N}}_{finite}$, $F(s)$ is a finite integer?
      - Look at the ASCII table. Why might I have gotten nervous about using the decimal code instead of the hex code?

      ii. Let $\mathcal{R}_{[0,1)}$ denote the set of positive reals less than one. Let $2^{\mathcal{R}_{[0,1)}}_{finite}$ denote the set of finite subsets of $\mathcal{R}_{[0,1)}$. Let $\mathcal{R}$ denote the set of reals (possibly bigger than one). Prove that $2^{\mathcal{R}_{[0,1)}}_{finite}$ has cardinality at most that of $\mathcal{R}$ by defining a concrete function $F(s) = x_s$ mapping each finite subsets s of the positive reals less than one to a unique real x_s. Be as explicit as you can. Hint: Interweave the bits. Be sure (but don't prove) that for your construction, for every $s \in 2^{\mathcal{N}}_{finite}$, $F(s)$ is valid real number and that if s and $s' \in 2^{\mathcal{R}}_{finite}$ are different then $F(s) \neq F(s')$.

   (b) A Hierarchy of infinities.

      i. Prove that for every set U, the cardinality of 2^U is strictly bigger than that of U, i.e. $|2^U| > |U|$. In our third definition of $|2^U| \leq |U|$, we argued that if each object $u \in U$ is able to hit at most one element $F^{-1}(u) = s \in 2^U$ and this process manages to hit every element $s \in 2^U$, then it follows that $|2^U| \leq |U|$. Conversely, we prove $|2^U| > |U|$, by proving that $\forall$ inverse functions F^{-1} from U ideally to 2^U, $\exists s_{new} \in 2^U, \forall u \in U, F^{-1}(u) \neq s_{new}$.

      Your proof should use the first order logic game between the adversary and the prover. Note unlike the proof that $|R| > |N|$, the set U might not be countable and hence can't be listed and hence the diagonal can be visualized.

      Hint: Woody Allen once said that he did not want to be a member of any club that would have him as a member. In this spirit, for each person, put him in the heaven club iff he is not in the club that is mapped to him.

ii. We can use the previous theorem that for every set U, $|2^U| > |U|$ to get many great results. For example, if U is the set of natural numbers $\mathcal{N}$, then we get that $|2^{\mathcal{N}}| > |\mathcal{N}|$ giving that the set $2^{\mathcal{N}}$ of subsets of $\mathcal{N}$ is uncountable. (We did not prove it, but $2^{\mathcal{N}}$ has the same cardinality as the reals $\mathcal{R}$.) As a second example, let U be the set of reals $\mathcal{R}$, then we get that $|2^{\mathcal{R}}| > |\mathcal{R}|$ giving that the set $2^{\mathcal{R}}$ of subsets of $\mathcal{R}$ is a bigger infinity than the number of reals. Prove by that there are a hierarchy of an infinite number of different sizes of infinity.

3. Computable and Describable Reals

   (a) A real number x is said to be *computable* if there is a Java program that on input zero prints out the decimal representation of x from left to right. Note that at no point in time will all of the digits of x be printed out, but for each digit of x, there will be an eventual time at which this digit will be printed out. Use the words taught in class to prove in one sentence whether or not all real numbers computable.

   (b) A real number x is said to be *describable* if it can be unambiguously denoted by a finite piece of English text. For example, $x = 2$ is described as "Two" and $x = \Pi$ as "The area of a circle of radius one." Use the words taught in class to prove in one sentence whether or not all real numbers describable.

   (c) Prove that every computable real is also describable.

   (d) Prove whether or not there a real number that can be described, but not computed? "Let x be the smallest real number that is not computable" is not a valid answer for the same real that "Let x be the smallest real number that is bigger than zero" is ill defined. Hint: Use a diagonalization proof.

4. `Not recursive enumerable.`

```
A computational problem is recursive enumerable if there is a program
such that on every yes instance the program halts with the answer yes
and on every no instance it can either halt with a no or can run
forever.  The halting problem is recursively enumerable because on yes
instances <x,I> the program phi_x does halt on input I and hence our
program for the halting problem when simulating it can halt with a
yes.  On the other hand, the not-Halting problem is not r.e. because
here on yes instances <x,I> the program phi_x does NOT halt on input I
and hence our program for the not-Halting problem when simulating it
must run for ever to know. Hence we prove that problem P is not
r.e. by the reduction not-Halting <= P.  It is key that in this
reduction that we do not switch the roles of yes and no, i.e. when the
oracle says yes, we say yes too and when the oracle says no we say no
too.

1a) A = { x : phi_x outputs exactly 5 distinct elements }
              eg for every input either outputs {1,2,3,4,5} or runs
              forever.
This problem combines Halting and non-Halting because when we say yes
on instance x when \phi_x halts on I with output 1 and we say yes when
```

\phi_x does not halt on I. This means that A is neither r.e. or co-r.e.

```
We will prove that not-Halting <= A.
Assume we have a quick algorithm for A.
We design an Alg for Halting as follows.
   Let <x,I> be the input to the Halting.
   Let phi_x' be the program that takes I' as input
      if I' in {1,2,3,4,5} then phi_x' halts with output I'.
      else phi_x' simulates phi_x on I.
           then phi_x' halts and outputs 6.
   We give x' to our algorithm for A.
   If it says yes we say yes and if it says no than we say no.
We now prove that this algorithm solves not-Halting.
If phi_x does not halt on I,
   then phi_x' outputs {1,2,3,4,5} on {1,2,3,4,5}
   and other wise runs forever.
   Hence, x' is in A and hence the oracle and we say yes.
If phi_x does halt on I
   then phi_x' outputs value 6 on other inputs.
   Hence, x' is not in A and hence the oracle and we say no.
This completes the reduction.
We know that not-Halting is neither recursive enumerable or
recursive. Hence neither is A.
```

```
1b) D = { x : phi_x is the characteristic function of some set}
       = { x : exists a set S such that phi_x(I)=1 I in S
                                        phi_x(I)=0 I not in S.
       = { x : phi_x halts on every input and has range {0,1} }
```

The difficulty in proving that D is not recursively enumerable is that the halting problem is recursively enumerable because on yes instances <x,I> the program phi_x does halt on input I and hence our program for the halting problem when simulating it can halt with a yes.
More over, if D asked whether phi_x halted on say the first million inputs then this would also be r.e. because if the answer was yes, the program for D could wait until all million computations halted. But D asks whether an infinite number of computations halt. Suppose on the t^th input the computation runs for t time steps. Then a program for D would have to run forever to check that each of these computations halted. This is the motivation for the following reduction.
Note that the not-Halting problem is not r.e. because here on yes instances <x,I> the program phi_x does NOT halt on input I and hence our program for the not-Halting problem when simulating it must run for ever to know. Hence we prove that D is not r.e. by the reduction
 not-Halting <= D

```
We will prove that not-Halting <= D.
Assume we have a quick algorithm for D.
We design an Alg for Halting as follows.
   Let <x,I> be the input to the Halting.
   Let x' be the program that takes t as input
      if phi_x(t) runs at least t steps then
*        x' halts and outputs 1.
      else x' halts and outputs 6.
   We give x' to our algorithm for D.
   If it says yes we say yes and if it says no than we say no.
We now prove that this algorithm solves not-Halting.
If phi_x does not halt on I, then it run at least t steps for every t.
   then x' for each input t halts and outputs 1.
   Hence, x' is in D and hence the oracle and we say yes.
If phi_x does halt on I after t time steps
   then x' outputs value 6 on input t+1.
   Hence, x' is not in D and hence the oracle and we say no.
This completes the reduction.
We know that not-Halting is neither recursive enumerable or
recursive. Hence neither is D.

It is a little interesting that that this reduction for D also works
for A but not visa versa.
1a) A = { x : phi_x outputs exactly 5 distinct elements }
The proof can be exactly the same as for 1b
except the line marked * is changed to
*        if t in {1,2,3,4,5} then x' halts and outputs t
         else x' halts and outputs 5.
In this way if phi_x does not halt on I,
then x' halts on every input and its range is {1,2,3,4,5}
else x' also outputs 6.
We know that not Halting is neither recursive enumerable or
recursive. Hence neither is A.
```

5. Undecidability Threshold of $\forall x M(x) = 0$ for Really Simple Machines M.

The Halting Problem is undecidable (uncomputable) because the given Turing Machine M might run arbitrarily long on the one given input x. The statement of Rice's Theorem talks about the requirement for each of M's instances x, but its proof still relies only on the length of the computation on a single instance. But want if we force the machine to be a linear time, log space, read once automata, but we want to know what this machine does on every instance, for example $\forall x M(x) = 0$ or $\forall x, y, M(x, y) = M(y, x)$. We prove that this too is undecidable if M is allowed $(1+\epsilon)n$ time and $(1+o(1)) \log n$ space but is linear time decidable if it is allowed only $n+\mathcal{O}(1)$ time or only $(1-o(1)) \log n$ space.

Def^n: Define a Deterministic t time s space Automata (DA(t, s)) A to be similar to a Deterministic

Finite State Automata (DFA) except that it uses t time and s space. It is a two binary tape Turing Machine. The input is read in one pass on the first tape, i.e. the head never goes left or writes. The second tape is a work tape that can only use $t(n)$ time and $s(n)$ cells on instances of length $|x| = n$.

Thm 1: If is undecidable whether or not a DA$((1+\epsilon)n, (1+o(1)) \log n)$ A outputs zero on every instance x.

Thm 2: If can be decided in linear time whether a DA$(n+\mathcal{O}(1), \infty)$ A outputs zero on every instance x.

Thm 3: If can be decided in linear time whether a DA$((\infty, (1-o(1)) \log n)$ A outputs zero on every instance x.

Intuition: This problem is hard because x can be arbitrarily long. We do a reduction to the halting problem, which is hard because M's computation might be arbitrarily long. Hence, we have x encode the length of M's computation.

Def^n: Given a Turing Machine M, define A_M to be the following DA$((1+\epsilon')n, (1+o(1)) \log n)$. On instance x, A_M first counts in binary every $\frac{1}{\epsilon}$ bit of x on its first tape. This takes $(1+2\epsilon)n$ time and $\log(\frac{1}{\epsilon}n)$ space. Next A_M counts the $\log(\frac{1}{\epsilon}n)$ bits used on it second tape. It then knows the value $\log\log(\frac{1}{\epsilon}n)$ because this is how much tape it uses for the second count. Then A_M simulates M on the zero instance for this number of time steps. If it halts during this time, A_M outputs one, otherwise it outputs zero. These last two steps each take $o(1)n$ time and $o(1)\log(n)$ space.

Proof of Thm 1: Clearly M halts on the zero instance if and only if it is not true that A_M outputs zero on every instance x. The halting problem is undecidable and hence so is this problem. ■

(**Commutablity**: The same proof works if the problem is not $\forall x A_M(x) = 0$, but is $\forall x, y, A_M(x, y) = A_M(y, x)$. A_M will be designed at before using it's input x and ignoring its input y. If M does not halt, then $\forall x, y, A_M(x, y) = 0 = A_M(y, x)$. If M does halt then there is some sufficiently large instance x and sufficiently small instance y such that $A_M(x, y) = 1$ and $A_M(y, x) = 0$.)

Lemma: If a DA A uses at most $(1 - o(1)) \log n$ space, then it uses at most $O(1)$ space and hence is a DFA.

Proof: Suppose it uses at most $(1-o(1)) \log n$ space and has q states in it's finite automata. Then there must be a constant s such that A uses at most s space on every an instance x' of length $n' = q2^s - 1$. With s space on the second tape, A has at most $q2^s$ configurations. Hence, for each such instance x', there must have been at least two times as it moved its head right n' times on the first tape, at which it was in the exact same configuration. Hence, on an arbitrarily long instance x extending x', A must cycle and hence use at almost $s = O(1)$ space. ■

Lemma: If a DA A uses at most $n+O(1)$ time, then it uses at most $O(1)$ space and hence is a DFA.

Proof: If it uses at most $n+O(1)$ time, then there must be a constant s such that for each instance x' of length $n' = q2^s - 1$, either A devotes at most s time and hence s space on the second tape or A does not spend the required n' time to read to the end of x' on its input tape. Note that on different x', A could make a different choice here. In the first case, A cycles as before. In the second case, A does not know about the end of its input. Either way, such an x' can be extended arbitrarily without A using more than $O(1)$ space. ■

Lemma: It can be decided in linear time whether or not a DFA A outputs zero on every instance x.

Proof of Lemma and of Theorem 2 and 3: Given an n bit description of a DFA A, consider its transition graph through its at most n transition edges. It outputs a nonzero on some instance iff there is a path in this graph from its start state to a state outputting such a value. This can be determined in $\mathcal{O}(n)$ time using depth first search. ■